CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Recap

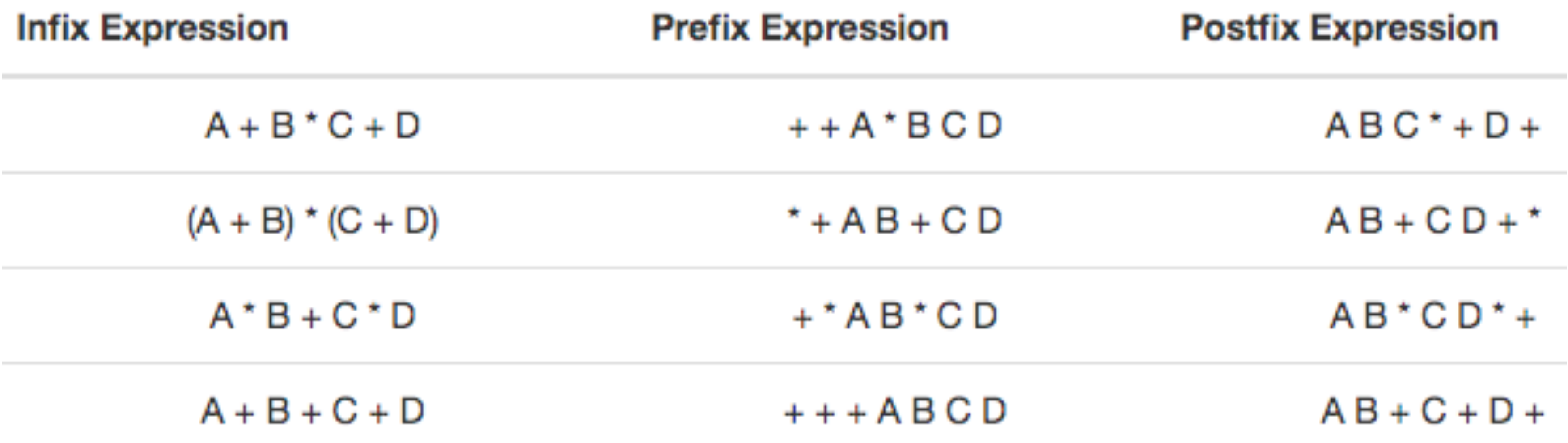
- The data structure of queue
- Implementation of Queue using array
- The application on Queue

Learning Topics

- Evaluating postfix arithmetic expression
- The Priority Queue data structure
- Applications on Priority Queue
- Implementation of Priority Queue using Ordered Array.
- [Optional] convert infix to postfix

More Applications on Stack

- Evaluation postfix expressions.



Infix Expression	Prefix Expression	Postfix Expression
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

- Please refer to the attached sample codes.

Priority Queues

- Priority Queue is an extension of the simple queue with the following properties:
  - Each element has a priority associated with it.
  - Each time when dequeuing elements from the priority queue, it is not longer the FIFO (first-in-first-out) policy, however, the one with the highest priority will be dequeued out.
  - If two elements have the same priority, they are served according to their enqueued order in the queue.

Priority Queues

- “Priority” has different definitions:
  - We can have *minimum priority queue*, in which, the smaller numbers have higher priority.
  - We can have *maximum priority queue*, in which the larger number have higher priority.
- Applications:
  - In Operating Systems to determine the scheduling order of processes.
  - In graph algorithm: minimum spanning tree, shortest paths.

Priority Queue

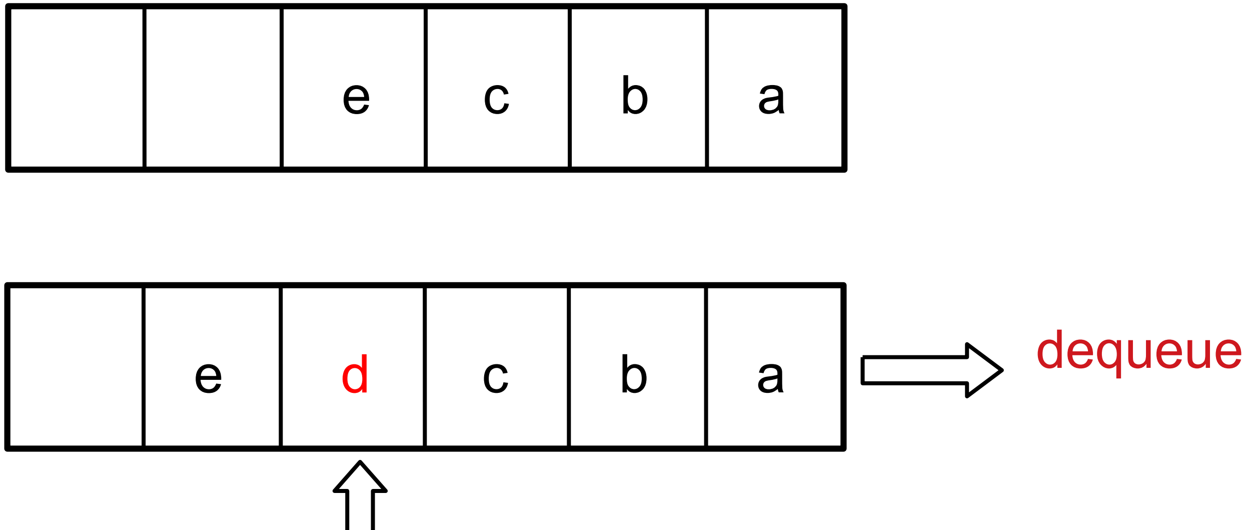
- Please check on the attached codes.

Priority Queues: implementation

There are different ways to implement Priority Queue. One of them is using ordered array, in which,

- To dequeue, we always remove from the front.
- To enqueue, we no longer always add to the rear:
  - We need to keep the order. So, add to the right place.
- Assume element with smallest key has the highest priority.

enqueue(d)



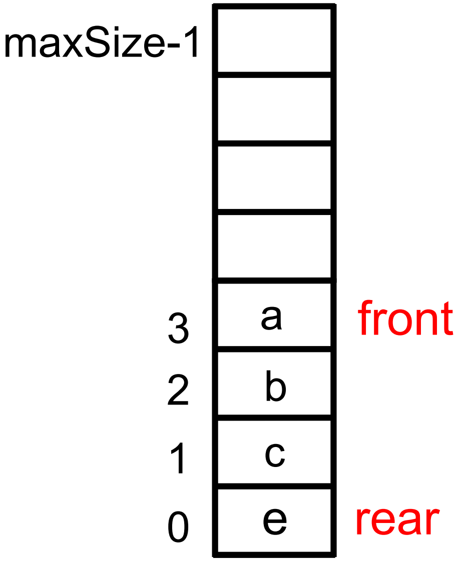
		e	c	b	a

	e	d	c	b	a

dequeue

Priority Queue-Implementation

- Can use an array to hold elements
- Fields needed
  - maxSize (capacity of queue)
  - queArray (array of elements in queue)
  - front (index of the front)
  - rear (index of the rear)
  - nItems(number of elements or size)

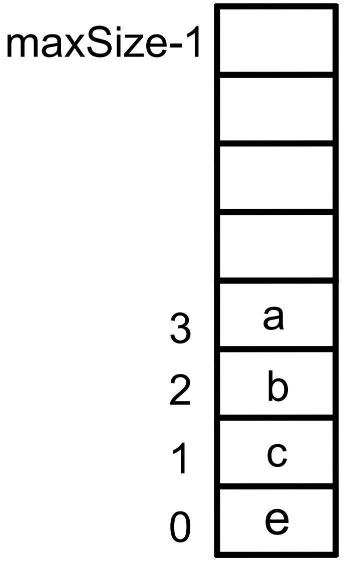


enqueue(d)

dequeue()

Priority Queue-Implementation

- Methods needed
  - enqueue(element)
  - dequeue()
  - peek()
  - size()
  - isEmpty()
  - isFull()



Priority Queue Class

```
class PriorityQ
{
        // array in sorted order, from max at 0 to min at size-1
        private int maxSize;
        private long[] queArray;
        private int nItems;
//-------------------------------------------------------------
        public PriorityQ(int s) // constructor
        {
                maxSize = s;
                queArray = new long[maxSize];
                nItems = 0;
        }
```

Priority Queue Class

```
public void enqueue(long item) // insert item while keeping the order
{
        if (isFull())
                throw new IllegalStateException("Queue is full");

        int i=nItems-1;
        while(i>=0 && queArray[i]<item)
        {
                queArray [i+1] = queArray [i]; //shift up
                i--; //go down
        }

        queArray [i+1]=item; //insert the item
        nItems++; //increment the number of items
}
```

Priority Queue Class

```
public long dequeue() // remove the highest priority element
{
            if (isEmpty())
                        throw new IllegalStateException("Queue is empty");

            return queArray [--nItems];
}
//--------------------------------------------------------------------
public long peek() // peek at highest priority item
{
            if (isEmpty())
                        throw new IllegalStateException("Queue is empty");

            return queArray [nItems-1];
}
```

Priority Queue Class

```
public boolean isEmpty() // true if queue is empty
{
        return (nItems==0);
}
//-----------------------------------------------------------
public boolean isFull() // true if queue is full
{
        return (nItems==maxSize);
}
//-----------------------------------------------------------
public int size() // number of items in queue
{
        return nItems;
}
}
```

Priority Queue Demo Class

```
class PQDemo
{
        public static void main(String[] args)
        {
                PriorityQ myPQ = new PriorityQ(4);

                myPQ.enqueue(4);
                myPQ.enqueue(2);
                myPQ.enqueue(3);
                myPQ.enqueue(1);

                while(!myPQ.isEmpty())
                {
                        System.out.println(myPQ.dequeue());
                }
        }
}
```

Parsing Arithmetic Expressions

- How do we evaluate arithmetic expressions: 2+3 or 2*(3+4)
  - *Infix* notation
- Easier to convert the expression into a *postfix* notation, then evaluate the postfix expression.
  - 23+ or 234+*
- Examples:
  - 1+2-3 → 12+3-
  - 1+2*3 → 123*+
  - 1*2 + 3*4 → 12*34*+

Convert infix to postfix

- **(For conversion from infix to postfix, read chapter 4 pages 149-166)**

- Examples:
  - 1+2-3 →12+3-
  - 1+2*3 → 123*+
  - 1*2 + 3*4 → 12*34*+